Close out and Final report for NASA Glenn Cooperative Agreement NCC3-751

Lapin Data Interchange Among Database, Analysis and Display Programs Using XML-Based Text Files

Summary

The purpose of grant NCC3-751 was to investigate and evaluate the interchange of application-specific data among multiple programs each carrying out part of the analysis and design task. This has been carried out previously by creating a custom program to read data produced by one application and then write that data to a file whose format is specific to the second application that needs all or part of that data. In this investigation, data of interest is described using the XML markup language that allows the data to be stored in a text-string. Software to transform output data of a task into an XML-string and software to read an XML string and extract all or a portion of the data needed for another application is used to link two independent applications together as part of an overall design effort.

This approach was initially used with a standard analysis program, Lapin, along with standard applications a standard spreadsheet program, a relational database program, and a conventional dialog and display program to demonstrate the successful sharing of data among independent programs. See "Engineering Analysis Using a Web-Based Protocol" by J.D. Schoeffler and R.W. Claus, NASA TM-2002-211981, October 2002.

Most of the effort beyond that demonstration has been concentrated on the inclusion of more complex display programs. Specifically, a custom-written windowing program organized around dialogs to control the interactions have been combined with an independent CAD program (Open Cascade) that supports sophisticated display of CAD elements such as lines, spline curves, and surfaces and turbine-blade data produced by an independent blade design program (UDO300).

Use of the UDO300 as an Independent Application

The UDO300 turbine blade design program takes nominal parameter values as inputs, designs a turbine blade, performs CFD analysis of the resulting blade, and outputs a specification of the resulting blade. This research uses the result of the UDO300 design as the initial turbine blade that is to be interactively repetitively modified and analyzed for performance.

UDO300 produces application specific files with blade parameters and data in text format. The most important data consists of the specification of a series of cross sections of the blade. Modification of the UDO300 design consists of changes to these cross sections. Analysis of such changes must be done by other application programs.

Interfacing to the UDO300 program then consisted of the design of a set of C++ classes supporting the data to be extracted along with a specification of a general XML specification of that data. The UDO300 classes were:

1. Class MultiUDOCrossSections
2. Class UDOCrossSection

Class MultiUDOCrossSections contains various parameter data produced by the design as well as a list of the individual cross section data. An object of class UDOCrossSection encapsulates the data of an individual cross section and specifies the cross section as three sets of 3 dimensional points (usually: 89 points on each of the suction and pressure curves and 29 points on the leading-edge curve).

The major task of class MultiUDOCrossSections is to read the UDO300-specific files produced by a design, and convert the data to XML strings. From the XML, the data was then converted to the (arbitrary) form required by the MultiUDOCrossSection object and the multiple UDOCrossSection objects (one per cross section produced).

XML Support for Interchange of UDO300 Data

Based upon a definition of XML data appropriate to general turbine blade analysis produced by TransenData Corp. under another NASA contract, a set of XML classes were created so as to allow the produced blade data to be stored as a tagged XML string. The classes are:

1. Class MultiGridNoValXml
2. Class MultiGridNoValSet
3. Class MultiGridNoVal
4. Class GridNoVal
5. Class GridNoValSet
6. Class GridNoValBlock
7. Class IJK
8. Class IJKRange

For a blade defined by a set of cross sections there is a single MultiGridNoVal object. The MultiGridNoVal contains one GridNoValBlock per cross section. The GridNoValBlock defines one cross section. It defines one IJKRange for all curves in that cross section. It defines a set of GridNoVal objects for that cross section, one for each point on each curve of that cross section.

The IJKRange has a single I parameter value and J parameters Jmin = 1 and Jmax = number of curves for this cross section (e.g., suction, pressure, and leading edge). It's K parameter has Kmin = 1, and Kmax = the maximum number of points per curve (the leading edge curve has a different number of points from the top and bottom curves).

Each GridNoVal corresponds to a single point on one of the curves of a cross section. It contains one IJK value (the parameterized index) and one Coordinate that contains the xyz values for that point on the curve.

Note that a GridNoValBlock defines multiple curves. It is the J parameter of the point (GridNoVal object defining the point) that specifies which curve.

The "NoVal" portion of each class indicates that this set of classes is a specialization of the original XML set that allowed any number of values to be associated with each point on each cross section (e.g., temperature, pressure, ...). Use of the UDO300 data (that does contain such data) in this program was restricted to the cross section specification data.

A key class is class MultiGridNoValXml that contains methods to generate the XML string from the UDO300 output files and to generate the MultiUDOCrossSections and UDOCrossSection objects from the XML string.

General Independent Blade Analysis Programs

Analysis of a turbine blade normally proceeds through a sequence of blades, each of which is a modification of one of the previous blades. The choice of the designer is carried out through the custom interactive design dialog program (below) but the analysis itself is supported mainly through the following set of classes:

1. Class BladeDataSet—definition of the set of cross sections of the blade and control over all transformations to be made to the cross sections.

2. Class BladeData—the data associated with a single cross section.

3. Class CrossSectionParams—parameters of a cross section that may be modified by the designer.

Each blade created by the designer is represented by an object of class BladeDataSet that in turn contains one object of class BladeData and one object of class CrossSectionParams for each cross section. Cross section data is stored more appropriately in class BladeData as a signe list of points on the curve starting at the suction trailing edge through the leading edge and ending at the trailing edge of the pressure curve. This facilitates creation of single curves representing the entire cross section and this in turn allows a CAD program to generate the overall surface of the blade for both analysis and display.

The CrossSectionParams object contains parameters of a cross section of interest to the designer. These include:

1. Geometric center of the cross section

2. Sweep of the cross section (offset relative to the geometric center of the first or reference cross section).

3. Rotation or twist angle of the cross section relative to the reference cross section.

4. The normal to the plane of the cross section.

The BladeDataSet object is responsible for the modifications to be made to the blade and the creation of the resulting new BladeDataSet. The interactive dialog program allows the user to control the creation of the set, changes to the parameters, as well as the display of each or all of the resulting blades. The first blade created is directly from the UDO300 data and consists of non-planar cross sections. All other blades have planar cross sections as are usually required by display and analysis programs.

The major modifications to a blade are done in methods of the BladeDataSet object of a specific blade. They include:

1. CreateBladeDataSetFromUDODataSet
2. CreateSingleBladeDataFromUDODataSet
3. TransformBladeSetToPlanarSections
4. TransformBladeSetToEquallySpacedParallelPlanarSections
5. TransformTranslatedBladeSetToBSplines
6. TransformBladeSetInPlaneRotation(double RIPmin,double RIPmax);
7. TransformBladeSetWithNewSweep
8. ChangeNumberOfCrossSectionPoints

Each such created blade needs parameters supplied by the designer through the custom interactive design dialog program and produces a complete new blade that may be displayed or further modified. However, the actual manipulation of the cross section data and the generation of the 3 dimensional points of the new section requires sophisticated analysis capability normally found in a CAD library. It is very desirable to isolate the analysis from the choice of CAD program and this is done by using an entirely independent set of CAD classes and providing an independent interface to be used by the blade classes discussed in this section. That is, no code specific to the Open Cascade CAD library is used in these classes as discussed in the next section.

Open Cascade Data Interchange and Control

The Open Cascade CAD library is an open software free library that allows the manipulation of CAD objects (points, curves, surfaces) and includes sophisticated analysis aids such as finding intersections of curves and planes, lines and surfaces, etc. It also contains a display program that allows display and rotation of view of a surface. This permits viewing directly the affects of a modification to a blade and comparison between blades.

To make the analysis classes independent of the Open Cascade library, a set of interface functions were defined specifically to support the transformations discussed in the previous section. Then a separate implementation of these functions using the Open Cascade library was separately created.

This proved to be a significant advantage because all the complexity of the Open Cascade library including its definitions of classes, its many header files, and its complexity becomes isolated from the C++ code of this project. Thus it would be relatively straightforward to replace Open Cascade analysis with that of another library or separate analysis program. Of course, eliminating the Open Cascade library would require the use of another display program since the Open Cascade display is used in this project.

The Custom Interactive Design Dialog Program

All the classes represent infrastructure that is useful to a designer. To take advantage of this infrastructure, an interactive design program that permits the designer to see selected data, modify selected data, create and view new blades, and generate XML output data to be used in external analysis programs is necessary.

A criterion for such an interactive design program is that it must be easy to create and modify without requiring structural changes to the infrastructure discussed in previous sections. To this end, a dialog-based program using Microsoft Windows MFC dialog classes was used. The most important characteristic of this choice is the ease with which individual dialogs may be created and added to the dialog program for specific design changes of interest.

Examples of dialogs used include the following:

1. Blade Set dialog—a master dialog that displays the group of blades that have been created thus far, and the alternative kinds of changes that a designer might make (e.g., twist the blade, change the blade sweep, etc.)

2. Cross section twist and offset dialog—a stand-alone dialog that is raised by a choice on the Blade Set dialog that displays the rotation and sweep of each cross section and permits the designer to modify these values and create a new blade set.

3. Simple dialogs that solicit additional data from the designer such as the number of cross sections desired in a new design, the number of points desired in the individual cross sections, etc.

Conclusions and Future Work

The separation of the components of the resulting design program makes it quite straightforward to modify the individual components. The change of the CAD program used is a good example of this. The ease with which new dialogs may be added to support new modifications to a blade is another.

To further demonstrate the ease of data interchange and the flelxibility and practicality of the approach used in this grant, two more steps should be taken. They are the following.

1. The XML of the modified blade should be used to generate input data appropriate to another completely independent analysis program. This would require also defining XML data for the output produced by this program and the conversion to the XML input data used to define blades in the current work. This would permit the demonstration of round-trip analysis and design: starting from an initial blade designed using the UDO300 program, analysis and modification and generation of input data to the second analysis program, recovery of output data from that

program in the form of an XML string that is then used to to specify further changes, etc.

2. The analysis/design program created under this grant using groups of independent classes that are easily extended as discussed above. However, the addition of something (like the use of the second analysis program above) requires the addition of classes to the program and the rebuilding of the entire program. It would be more desirable to componentize this program to facilitate ease of adding additional flexibility. Of particular interest would be to recreate the parts of this program as "web-based components" that may independently added and accessed without major rebuilding of existing computer programs. This is the trend in commercial application development today and is an attractive approach to make the interchange of data among independent analysis and design programs easier and more effective.